Performance Testing and Performance Improvement Methods for Communicating Systems

Levente Erős

MSc. in Technical Informatics

Department of Telecommunications and Media Informatics
Doctoral School of Informatics
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics

Summary of PhD. Dissertation

Supervised by:

Dr. Tibor Csöndes

Honorary Associate Professor

Department of Telecommunications and Media Informatics
Faculty of Electrical Engineering and Informatics
Budapest University of Technology and Economics

Budapest, Hungary

2012.

1 Introduction

Testing plays a vital role in the development of a communicating system implementing a certain communication protocol. After the implementation phase, the developed system is regarded as a black box and different kinds of tests are executed against it, in order to check whether it corresponds to its different kinds of requirements.

In the field of telecommunications, a conformance test checks whether the system under test (SUT) implements the communication protocol that it should implement according to its conformance requirements, while a performance measures different performance characteristics of the SUT.

Conformance testing has taken a long journey from fully manual, ad-hoc testing to model based testing [1, 2] (see Figure 1) throughout the years, and developed an evolved theoretical background including formal description methods for modeling the functional behavior of the SUT [3–5], and semi-automatic conformance testing methods [6–13], as well as test suite management methods [14–19].

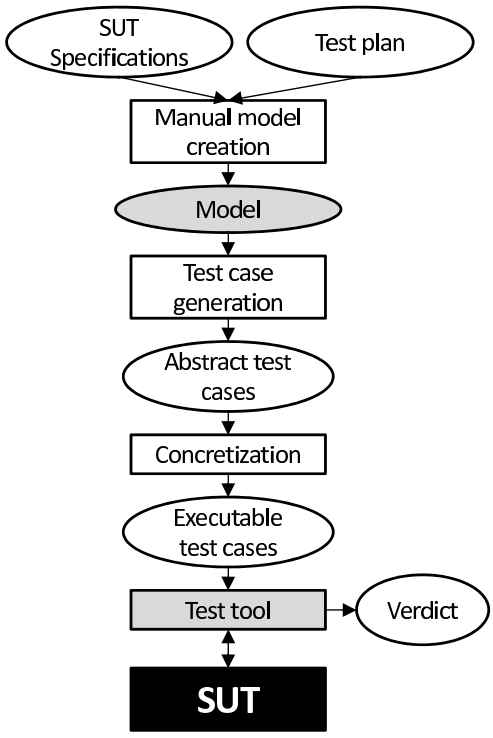

Figure 1: Model-based test design

Contrarily to conformance testing, black-box performance testing lacks an evolved theoretical background. While there are papers presenting techniques for modeling the performance of a communicating system [20–23], and also papers dealing with issues of performance test execution [24–26], performance tests of communicating systems are designed manually, in an ad-hoc way, without any theoretical background (see Figure 2). Thus, one of my goals was to create an automatic performance testing method.

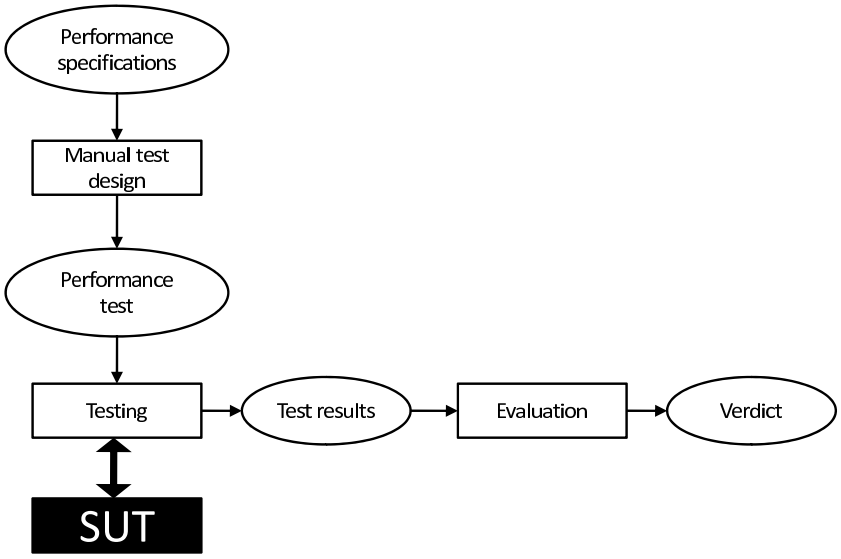

Figure 2: Performance testing

Besides the lack of a theoretical background, another problem of performance testing is how to exploit the capacities of the performance test environment. This latter is a real problem, since while the SUT is a system optimized for a specific purpose, the hardware used in the test environment is universal, and is reused for executing different performance tests. However, the test environment has to generate a relatively high load towards the SUT. To solve this problem, multiple hosts are used in the test environment, which together, are capable of generating the necessary load. In the industry, load generating software entities are used for generating stress towards the SUT. These load generators are executed on the hosts of the test environment (testing hosts). In order for the test environment to be able to efficiently generate the necessary load towards the SUT, methods are needed, which assign the load generators to the testing hosts, with the aim of exploiting the capacities of the test environment as much as possible.

2 Research Objectives

The objective of my research was to solve the earlier mentioned open problems of black-box performance testing of communicating systems.

In the first part of my thesis, my research objective is to create methods for distributing the load generator entities among the testing hosts of the test environment, with the aim of maximizing the average utilization of testing hosts.

In the second part of my thesis, my goal is to create an efficient, automatic performance testing method, which is capable of determining whether the SUT is capable of processing the maximal number of request messages it has to serve within a second while serving a given number of users simultaneously.

In the third part of my thesis, my goal is to create methods for correcting the performance of the SUT at minimal cost, once it has failed the performance test executed on it according to the performance testing method presented in the second part.

3 Methodology

In the first and third thesis groups, for proving the complexity of the solved problems, I used analytical methods. I have formulated the problems as integer linear programs and proposed heuristic algorithms for solving them. The performance of these methods has been investigated by simulations.

For solving the problem dealt with in the second thesis group, I have defined a mathematical model and two methods. I used analytical methods for proving the correctness of one of the methods, while the correctness of the other method has been provenFthe ob by experiments.

4 New Results

4.1 Load Distribution in a Performance Testing Environment

As mentioned earlier, during a performance test, the test environment – composed of universal hardware – has to generate relatively high load towards the SUT – which is a piece of hardware optimized for a specific purpose. This is achieved by using

multiple testing hosts (THs from now on) in the test environment which, together are capable of generating this load.

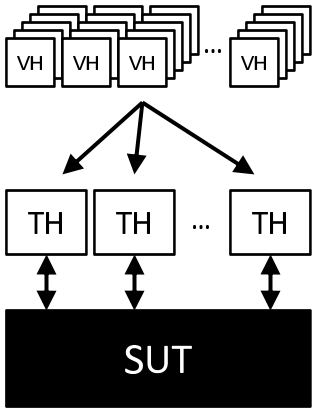

Figure 3: Assigning THs to VHs

The load generating software entities or virtual hosts (VHs from now on) used for generating stress towards the SUT, run on THs and usually, the number of VHs is larger than the number of THs. Each TH has a total capacity, while each VH has a required capacity, and a VH can only be assigned to a TH if, for the whole duration of the execution of the VH, the free capacity of the TH is greater than or equal to the required capacity of the VH. Each VH has to be either assigned to a TH or dropped (Figure 3). The objective of the VH assignment is to maximize the average utilization of THs (that is, to maximize the load generated by the test environment). The problem described above will be referred to as the load distribution problem.

Thesis 1 *I have proven the NP-completeness of the load distribution problem. I have formulated the problem as an integer linear program, and gave a method, which solves the problem using a series of integer linear programs. Furthermore, I have given a heuristic algorithm for solving the load distribution problem.*

I have defined the load distribution problem on a discrete time axis composed of atomic time slots. I have formalized the problem as follows:

Given are the set of THs $\mathcal{TH} = \{TH_i\}$ and the set of VHs $\mathcal{VH} = \{VH_i\}$. Each TH has one attribute $TH_i = (TC_i)$, where TC_i is the total capacity of TH_i. Each VH has three attributes $VH_i = (ST_i, RT_i, C_i)$, where ST_i is the starting time of VH_i (i.e. the number of the time slot in which VH_i starts), RT_i is the execution

time of VH_i (i.e. the number of time slots that the execution of VH_i takes), and C_i is the required capacity of VH_i.

The problem to be solved is as follows: For each $VH_i \in \mathcal{VH}$, choose the value of assignment function $\sigma \in \mathcal{VH} \rightarrow \mathcal{TH}$ from domain $\mathcal{D} = \mathcal{TH} \bigcup \{\emptyset\}$ such that, Formulas 1 and 2 below are true. Choosing TH_j as the value of $\sigma(VH_i)$ corresponds to assigning VH_i to TH_j, while choosing $\emptyset$ as the value of $\sigma(VH_i)$ corresponds to dropping VH_i. In Formula 1, U is a lower limit for the total TH capacity. If u is the total TH utilization (used TH capacity to total TH capacity ratio), then $U = u \sum_{i=1}^{|\mathcal{TH}|} TC_i t_{max}$. In the formula and the rest of the section, t_{max} denotes the last time slot ($t_{max} = \max_{k:\, VH_k \in \mathcal{VH}} (ST_k + RT_k - 1)$).

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k + RT_k - 1}} C_k \geq U \tag{1}$$

$$\forall (i : TH_i \in \mathcal{TH}) : \forall (j = 1, \ldots, t_{max}) : \sum_{\substack{k:\sigma(VH_k)=TH_i \wedge \\ \wedge ST_k \leq j \wedge \\ \wedge j \leq ST_k + RT_k - 1}} C_k \leq TC_i \tag{2}$$

Formula 1 states that the total utilization of THs should be above the lower limit U, while Formula 2 expresses that for each TH_i, in each time slot, the aggregated capacity of VHs running on TH_i, must be lower than or equal to the total capacity of TH_i.

Thesis 1.1 *[J1, C1] I have proven the NP-completeness of the load distribution problem, and formulated it as an integer linear program. I have given a heuristic method for solving the load distribution problem. The algorithm splits up the time axis into time windows, and solves the load distribution problem as a series of integer linear programs, each of which is formulated for a given time window.*

I have proven the NP-completeness of the load distribution problem by reducing an arbitrary instance of the knapsack problem with identical value and weight functions, which is an NP-complete problem [27]. The reduction can be found in Section 2.2 of my dissertation. Since the problem is NP-complete, in order to find its optimum, it has to be formulated as an integer linear program, which is going to be a binary linear program in this case.

Before formulating the load distribution problem as a binary linear program (BLP from now on), the boolean variable a_{kj} has to be defined (for the easier readability of the formulas) as follows:

$$a_{kj} = \begin{cases} 0 & \text{if } ST_k \leq j \wedge ST_k + RT_k - 1 \geq j \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

Furthermore, a new TH $TH_{|\mathcal{TH}|+1}$ has to be introduced. Assigning VH_k to $TH_{|\mathcal{TH}|+1}$ represents dropping VH_k. The total capacity of this TH is infinite or technically, it is equal to the sum of the capacities of all VHs (in order for each VH to be able to be assigned to it), formally:

$$\begin{gathered} \mathcal{TH}' = \mathcal{TH} \bigcup \{TH_{|\mathcal{TH}|+1}\}, \text{ where} \\ TH_{|\mathcal{TH}|+1} = (TC_{|\mathcal{TH}|+1}) \text{ and} \\ TC_{|\mathcal{TH}|+1} = \sum_{k:\, VH_k \in \mathcal{VH}} C_k \end{gathered} \tag{4}$$

The BLP formulation of the load distribution problem is as follows. The unknown variables the values of which have to be found are variables s_{ki}. The value of s_{ki} is chosen to be 1 if VH_k gets assigned to TH_i, otherwise its value is 0.

Maximize:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=1}^{t_{max}} \sum_{k=1}^{|\mathcal{VH}|} a_{kj} s_{ki} C_k \tag{5}$$

Subject to:

$$\forall (i : TH_i \in \mathcal{TH}') : \forall (l : VH_l \in \mathcal{VH}) : \sum_{k=1}^{|\mathcal{VH}|} a_{kST_l} s_{ki} C_k \leq TC_i \tag{6}$$

$$\forall (k : VH_k \in \mathcal{VH}) : \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{7}$$

$$\forall (k : VH_k \in \mathcal{VH}) : \forall (i : TH_i \in \mathcal{TH}') : s_{ki} \in \{0, 1\} \tag{8}$$

As simulations of different scenarios have shown in Section 2.5 of my dissertation, in many cases it is not realistic to solve the above binary linear program, due to its huge running time. Thus, I have developed a heuristic algorithm for solving the problem as a series of binary linear programs.

The algorithm divides up the time axis into time windows of size W, and formulates a sub-problem for each time window. Formulas 9 to 12 formulate the load distribution problem for time window n. In the formulation, the value of S_k is a reference to the TH to which, VH_k was assigned before time window n. That is, S_k equals i, if and only if VH_k was assigned to TH_i before time window n. If VH_k starts after the time window preceding time window n, then S_k equals -1. If S_k equals $|\mathcal{TH}| + 1$, then VH_k is dropped.

Algorithm 1: ILP based heuristic solution of the load distribution problem

input : $\mathcal{TH}$, $\mathcal{VH}$, W
output: $\bigcup_{k:\, VH_k \in \mathcal{VH}} \{S_k\}$

```
1 foreach VH_k ∈ VH do
2 |   S_k := −1;
3 n := 1;
4 while (n − 1)W + 1 ≤ t_max do
5 |   Formulate and solve BLP_n;
6 |   foreach k : VH_k ∈ VH ∧ ST_k ≥ (n − 1)W + 1 ∧ ST_k ≤ nW do
7 |   |   foreach i : TH_i ∈ TH do
8 |   |   |   if s_ki = 1 then
9 |   |   |   |   S_k := i
```

For the whole duration of the test, the VH assignment is carried out by Algorithm 1, which uses the following BLP formulation. The BLP defined by Formulas 9 to 12 for time window n, is denoted by BLP_n in the algorithm.

Maximize:

$$\sum_{i=1}^{|\mathcal{TH}|} \sum_{j=(n-1)W+1}^{nW} \sum_{\substack{k:\, VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kj} s_{ki} C_k \tag{9}$$

Subject to:

$$\begin{gathered}\forall(i:\ TH_i \in \mathcal{TH}'):\\ \forall(l:\ VH_l \in \mathcal{VH} \wedge ST_l \geq (n-1)W+1 \wedge ST_l \leq nW):\\ \sum_{\substack{k:\ VH_k \in \mathcal{VH} \wedge \\ \wedge ST_k \geq (n-1)W+1 \wedge \\ \wedge ST_k \leq nW}} a_{kST_l} s_{ki} C_k \leq TC_i - \sum_{\substack{k:\ VH_k \in \mathcal{VH} \wedge \\ \wedge S_k = i}} a_{kST_l} C_k\end{gathered} \tag{10}$$

$$\forall(k:\ VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W+1 \wedge ST_k \leq nW): \sum_{i=1}^{|\mathcal{TH}'|} s_{ki} = 1 \tag{11}$$

$$\begin{gathered}\forall(k:\ VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W+1 \wedge ST_k \leq nW):\\ \forall(i:\ TH_i \in \mathcal{TH}'): s_{ki} \in \{0,1\}\end{gathered} \tag{12}$$

Algorithm 1 gets sets $\mathcal{TH}$ and $\mathcal{VH}$ as its input, and outputs the S_k values belonging to each VH.

In lines 1 and 2, the algorithm initializes each S_k to initial value -1. From line 4, the algorithm runs iterations, one for each time window. Within an iteration, in line 5, the algorithm formulates and solves the BLP for the current time window. In lines 6 to 9, based on the calculated s_{ki} values of the BLP, the algorithm assigns the S_k value of each of those VHs, which were assigned to a TH (or dropped) in the current time window. By the end of the algorithm, each S_k value is known.

As simulations in Section 2.5 of my dissertation have shown, there are scenarios, in which not only the ILP formulating the original problem, but the ILP based heuristic algorithm cannot be solved due to its huge running time. For these scenarios thus, I have developed another heuristic algorithm, which is described in the following.

Thesis 1.2 *[J1, C1] I have proposed a bin packing based heuristic algorithm for solving the load distribution problem. The algorithm splits up the time axis into time windows of a given length, and considers and solves the load distribution problem as a series of bin packing-like problems, one for each time window.*

The main idea of the heuristic algorithm is that VHs the execution of which starts closely to each other, affect each other's assignability, just as the goods affect each other's assignability in the case of the bin packing problem [27]. After dividing up the time axis into time windows of size W, the assignment problem of VHs

starting in the same time window is similar to a bin packing problem. Thus, for VH assignment within the same time window, the heuristic algorithm uses a heuristic algorithm similar to the first fit descending (FFD) algorithm [28], which is a heuristic algorithm for bin packing.

Algorithm 2: Bin packing based heuristic solution for the load distribution problem

input : $\mathcal{TH}$, $\mathcal{VH}$, W

output: $\bigcup_{k:\, VH_k \in \mathcal{VH}} \{S_k\}$

1 **foreach** $VH_k \in \mathcal{VH}$ **do**
2 $\quad S_k := -1$;
3 $n := 1$;
4 **while** $(n-1)W + 1 \leq t_{max}$ **do**
5 $\quad \mathcal{VH}_n := \{\, VH_k | \, VH_k \in \mathcal{VH} \wedge ST_k \geq (n-1)W + 1 \wedge ST_k \leq nW\}$;
6 $\quad \mathbf{VH}_n \leftarrow$ sort $\mathcal{VH}_n$ by C_k descending; Return k;
7 $\quad k := 1$;
8 $\quad$ **while** $k \leq |\mathbf{VH}_n|$ **do**
9 $\quad\quad i := 1$;
10 $\quad\quad$ **while** $i \leq |\mathcal{TH}|$ **do**
11 $\quad\quad\quad$ **if** $\forall (j : a_{(\mathbf{VH}_n[k])j} = 1) : C_{\mathbf{VH}_n[k]} \leq TC_i - \sum_{l:\, VH_l \in \mathcal{VH} \wedge S_l = i} a_{lj} C_l$ **then**
12 $\quad\quad\quad\quad S_{\mathbf{VH}_n[k]} = i$;
13 $\quad\quad\quad\quad$ break;
14 $\quad\quad\quad i := i + 1$;
15 $\quad\quad$ **if** $S_k = -1$ **then**
16 $\quad\quad\quad S_k := |\mathcal{TH}| + 1$;
17 $\quad\quad k := k + 1$;
18 $\quad n := n + 1$;

Algorithm 2 shows the steps of the heuristic algorithm used for solving the load distribution problem. If in the algorithm, S_k equals $|\mathcal{TH}| + 1$ then VH_k is dropped.

After initializing each S_k to -1 in lines 1 and 2, the heuristics runs iterations, one for each time window. In line 5, the algorithm creates set $\mathcal{VH}_n$ from those VHs, which start in the current time frame. Then, from line 6 to 17 the heuristics assigns each VH from $\mathcal{VH}_n$ to a TH included in TH', using a heuristic algorithm similar to

the first fit descending algorithm, according to the following:

In line 6, the elements of $\mathcal{VH}_n$ are sorted by capacity in descending order and their references are put into vector $\mathbf{VH}_n$. Then for each VH_k for which, k is an element of $\mathbf{VH}_n$, starting from the VH_k element with the largest capacity, the algorithm finds the first TH from $\mathcal{TH}$, which has enough free capacity to execute VH_k that is, the first TH the free capacity of which is greater than or equal to the capacity of VH_k in each time slot in which VH_k is running. If the algorithm fails to find such a TH, then VH_k is assigned to $TH_{|\mathcal{TH}|+1}$ that is, VH_k is dropped.

In Section 2.5 of my dissertation, I have evaluated the time efficiency of and the maximal average TH-utilization achieved by the proposed heuristic methods and a greedy algorithm. According to the simulations, the proposed heuristic approaches are more efficient than the greedy load distribution algorithm in many scenarios.

Table 1 shows the comparison of the methods introduced in this section and the greedy algorithm.

Algorithm	Short description	Efficiency
Greedy	Assigns VHs to THs in the order of their starting times. Each VH is assigned to the first TH having enough free capacity for the whole time of execution of the VH	Time efficient, but achieves the lowest maximal average utilization
BLP based	Divides up the time axis into time windows and solves a BLP for each time window	Achieves a larger average TH-utilization with increasing window size, but unscalable due to NP-complete sub-problems
Bin packing based	Divides up the time axis into time windows and solves a bin packing-like problem in each time window	Reasonable running time, its maximal average TH-utilization is between those of the greedy and the BLP based methods

Table 1: Comparison of VH-assignment methods

4.2 A Model-Driven Performance Testing Method for Communicating Systems

As mentioned earlier, the field of black-box performance testing of communicating systems lacks an evolved theoretical background, including automatic methods for checking whether the SUT fulfills certain performance requirements. Black box performance tests are mainly designed in an ad-hoc way in the industry [29]. The main drawback of an ad-hoc method is the inaccuracy of the performance measurements it produces. To solve this problem, I have proposed an automatic, model-driven performance testing method. The presented method interacts with the SUT, and creates its formal performance model based on which, it automatically determines whether the SUT fulfills the performance requirement of serving CR_{usr} messages within a second while serving *usr* users.

CR_{usr} as a performance requirement is ambiguous. Thus, I interpreted this notion in two ways. CR_{usr} can be disambiguated as CWR_{usr}, which is the required number of messages that the SUT has to be able to serve within a second, in worst case. By worst case, I mean that the SUT has to be able to serve CWR_{usr} messages per second given any sequence of requests it receives from the users. CR_{usr} can also be disambiguated as CER_{usr}, which is the expected number of messages the SUT has to be able to serve within a second.

Thesis 2 *I have defined the Timed Communicating Finite Multistate Machine model (TCFMM) for representing some aspects of the performance of the System Under Test (SUT). I have given a method for calculating the worst-case number of requests that the SUT is capable of serving within a second. Furthermore, I have given a method for calculating the expected number of request messages that the SUT is capable of serving within a second.*

The proposed method is a model-driven performance testing method. The inputs of the method are the finite state machine (FSM) to which, the SUT corresponds, according to its conformance test, and the above mentioned conformance requirements. From this FSM and measurements performed on the SUT, the proposed method builds up a formal performance model of the SUT based on which, it analytically determines whether the SUT fulfills its performance requirements.

Thesis 2.1 *[J2, J5, C2] I have defined the Timed Communicating Finite Multistate Machine (TCFMM) model. The TCFMM is capable of representing the maximal*

number of request messages the SUT is able to process within a second while serving a given number of users. I have specified how to create the TCFMM model of the SUT based on the Finite State Machine (FSM) to which it corresponds and based on measurements performed on the SUT.

The Timed Communicating Finite Multistate Machine is defined in the following. In the definition, τ_0 denotes the time of the beginning of the execution.

Definition 1 *The Timed Communicating Finite Multistate Machine (TCFMM) is a 10-tuple:*

$$
\begin{aligned}
TCFMM = \; & (I, O, S, s_0, T, U, H, \delta, \chi, \sigma), \textit{ where} \\
1. \; & T \subseteq I \times O \times S \times S \times \mathbb{R}^+ \\
2. \; & \forall t_i, t_j \in T((t_i = (i_i, o_i, s_{from_i}, s_{to_i}, d_i) \wedge t_j = (i_j, o_j, s_{from_j}, s_{to_j}, d_j) \wedge \\
& s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j) \\
3. \; & \chi \in H \rightarrow U, \; \chi \textit{ is bijective} \\
4. \; & s_0 \in S \\
5. \; & \sigma \in \mathbb{R}^+ \times U \rightarrow S \cup \{\oslash\} \\
6. \; & \forall (u \in U) : \sigma(\tau_0, u) = s_0 \\
7. \; & \delta \in \mathbb{R}^+ \times H \times I \rightarrow S \times O \times \mathbb{R}^+ \\
8. \; & \forall (t_i \in T, h \in H, \tau \in \mathbb{R}^+) : \sigma(\tau, \chi(h)) = s_{from_i} \Rightarrow \delta(\tau, h, i_i) = (s_{to_i}, o_i, d_i) \\
& \textit{and invalid inputs are dropped.} \\
9. \; & \forall (\tau, h, i, s, o, d : \delta(\tau, h, i) = (s, o, d)) : \\
& (\forall (\phi : 0 < \phi < d) : \forall (u \in U - \{\chi(h)\}) : \\
& \mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1 \Rightarrow (\sigma(\tau + d, \chi(h)) = s \wedge \forall (\epsilon : 0 < \epsilon < d) : \\
& \sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge (\neg(\forall (\phi : 0 < \phi < d) : \forall (u \in U - \{\chi(h)\}) : \\
& \mathbf{P}(\sigma(\tau + \phi, u) = \oslash) = 1) \Rightarrow \exists (d' : d' < d) : \\
& (\sigma(\tau + d', \chi(h)) = s \wedge \forall (\epsilon : 0 < \epsilon < d') : \sigma(\tau + \epsilon, \chi(h)) = \oslash)) \wedge \\
& \wedge \exists (t_i \in T) : s_{from_i} = \sigma(\tau, \chi(h)) \wedge s_{to_i} = s \wedge i_i = i \wedge o_i = o \wedge d_i = d
\end{aligned}
$$

The proposed performance testing method creates the TCFMM model of the SUT. To create this model, first, its functional structure has to be built based on the Finite State Machine (FSM) to which, the SUT corresponds, according to the conformance test previously run on it. Let M'' denote the FSM to which the SUT corresponds. Following the FSM definition in [4], M'' is defined in the following way:

$M'' = (I'', O'', S'', \delta'', \lambda'')$, where
$\delta'' : S'' \times I'' \rightarrow S''$ is the state transition function
$\lambda'' : S'' \times I'' \rightarrow O''$ is the output function

Let $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$ denote the TCFMM, which is used for conducting the performance test. M is constructed in two steps from M''. In the first step, based on M'', a TCFMM $M' = (I', O', S', s'_0, T', U', H', \delta', \chi', \sigma')$ is created according to the following assignments:

- $S' = S''$
- $s'_0 =$ the initial state of M''
- $O' = O''$
- $I' = I''$
- $U' = \{u_i | i = 1, ..., usr\}$
- $H' = \{h_i | i = 1, ..., usr\}$
- $\forall (h_i \in H') : \chi'(h_i) = u_i$
- $T' = \{t_i = (i_i, o_i, s_{from_i}, s_{to_i}, \oslash) : \exists (i \in I'', s \in S'') : s_{from_i} = s \wedge s_{to_i} = \delta''(s, i) \wedge$
  $\forall t_j = (i_j, o_j, s_{from_j}, s_{to_j}, \oslash) : (s_{from_i} = s_{from_j} \wedge i_i = i_j) \Rightarrow t_i = t_j\}$

Functions σ' and δ' are derived from the above assignments, and from Constraints 5 to 9 of Definition 1. In the second step, M' is transformed to $M = (I, O, S, s_0, T, U, H, \delta, \chi, \sigma)$, according to the following assignments:

- $I = I'$
- $O = O'$
- $s_0 = s'_0$
- $U = U'$
- $H = H'$
- $\forall (h_i \in H) : \chi(h_i) = u_i$
- $T = \{t_i | \exists (t_j \in T') : (\exists (t_k \in T') : s_{from_k} = s_{to_j} \wedge s_{from_j} = s_{from_i} \wedge s_{to_j} = s_{to_i} \wedge$
  $\wedge i_j = i_i \wedge o_j = o_i \wedge d_j = d_i) \vee$
  $\vee (t_{to_i} = s_0 \wedge \exists (t_j \in T') : (\nexists (t_k \in T') : s_{from_k} = s_{to_j}) \wedge$
  $\wedge s_{from_i} = s_{from_j} \wedge i_i = i_j \wedge o_i = o_j \wedge d_i = d_j)\}$
- $S = \{s_i | s_i \in S' \wedge \exists (t_j \in T') : s_{from_j} = s_i\}$

Functions σ and δ are derived from the above assignments, and from Constraints 5 to 9 of Definition 1.

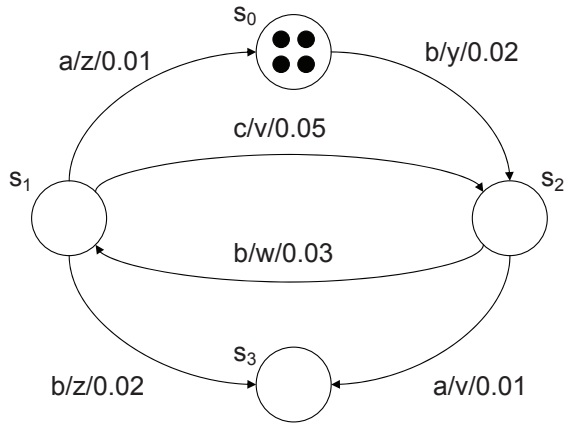

Figure 4: Graphical representation of the TCFMM

Figure 4 shows the graphical representation of a TCFMM. The transition parameters written on each transition are *input/output/delay*, respectively. All the tokens of the TCFMM reside in s_0.

During testing, the test environment emulates users towards the SUT and uses M for tracing the state changes of all the protocol instances (or server threads) running on the SUT. Placing *usr* tokens into M means that during the performance measurement, the test environment will emulate *usr* users. During testing, moving token u along transition t_i from state s_{from_i} to state s_{to_i} corresponds to the test environment sending input i_i to the SUT and then waiting to receive output o_i from the SUT, in the name of user h, where $\chi(h) = u$. When constructing M from M', all the transitions leading to states with no outgoing transitions (or sink states) in M', are redirected to s_0. This way, if a user emulated by the tester sends its last message to the SUT, it will reappear as a new user to be served, with its token residing at s_0. Thus, the SUT has to serve *usr* users in parallel, at all times.

In order to complete the TCFMM model representing the SUT, all the yet unknown transition delays have to be measured on the SUT. This is done as follows:

During testing, the test environment emulates *usr* users towards the SUT. No user entity can be idle during testing that is, upon receiving an output o_i from the SUT, the user entity has to send an input i_j to the SUT immediately, where $s_{to_i} = s_{from_j}$. The delay of each transition t_i equals the time elapsed between an emulated user sending i_i to the SUT and receiving o_i from the SUT. Each transition delay is

measured a predefined number of times, and these measurements are averaged to get the actual value of each d_i.

Thesis 2.2 *[J2, J5, C2] I have given a sufficient and necessary requirement of the SUT being capable of processing a given number of request messages within a second, in worst case. This requirement is defined on the TCFMM representing the SUT. Based on this sufficient and necessary requirement, I have given a method for calculating the number of requests the SUT is capable of seving within a second, in worst case.*

As proven in Section 3.5.1 of my dissertation, the sufficient and necessary requirement of a system processing CWR_{usr} messages per second in worst case is as follows:

$$\forall(c_i \in C) : \sum_{t_j \in c_i} d_j \leq \frac{|c_i|}{CWR_{usr}} \tag{13}$$

Based on the above sufficient and necessary requirement, the worst-case number of messages the SUT is able to process within a second can be calculated as follows:

$$CW_{usr} = \min_{c_j \in C} \left\{ \frac{|c_i|}{\sum\limits_{t_j \in c_i} d_j} \right\} \tag{14}$$

Thesis 2.3 *[J2, J5, C2] Based on the TCFMM model of the SUT, and state transition probabilities describing user behavior, I have given a method for calculating the expected number of requests that the SUT is capable of serving within a second.*

If the SUT corresponds to Formula 13, it is able to serve CWR_{usr} messages per second in worst case. However, the users communicating with the SUT in its latter real-life environment, might send an input message to the system more likely than another input message at a given state of execution. This behavior of users assigns different probabilities to transitions having the same originating state and consequently, the users might experience that the number of messages the SUT processes within a second is significantly higher than CW_{usr}. In the following, I show how to calculate CE_{usr}, which is the number of messages the system processes within a second, according to the experience of the users. CE_{usr} is always greater than or equal to CW_{usr}.

Let us assume that for a user h, token $\chi(h)$ resides at state s_{from_i}. Then the probability of t_i, p_i denotes the probability of user h sending i_i to the SUT (p_i is equal for each user). Let us furthermore define p_{kl} as follows:

$$p_{kl} = \sum_{i:t_i \in T \wedge s_{from_i} = s_k \wedge s_{to_i} = s_l} p_i \tag{15}$$

Thus, if $\chi(h) = s_{from_i} = s_k$, p_{kl} is the probability of user h sending to the SUT the input message of any transition leading from s_k to s_l. In other words, p_{kl} is the probablility of token $\chi(h)$ transferring from s_k to s_l (p_{kl} is equal for each user and token). Let furthermore z_i denote the *stationary state probability* of state s_i that is, the probability of token u transferring to state s_i *at any time* (z_i is equal for each token and user). To calculate CE_{usr}, we first have to calculate z_i for each state s_i from the following matrix equation, where $n = |S| - 1$ that is, the number of states in M minus one:

$$\boldsymbol{F}\boldsymbol{z} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \text{ where } \boldsymbol{z} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_n \end{bmatrix}, \text{ and } \boldsymbol{F} = \begin{bmatrix} p_{00} - 1 & p_{10} & \dots & p_{n0} \\ p_{01} & p_{11} - 1 & \dots & p_{n1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{0(n-1)} & p_{1(n-1)} & \dots & p_{n(n-1)} \\ 1 & 1 & \dots & 1 \end{bmatrix} \tag{16}$$

If $\det \boldsymbol{F} \neq 0$, the above matrix equation gives a definite solution for the z_i values [30]. Based on the z_i values, CE_{usr} is calculated as follows, where z_{from_i} is the stationary state probability of state s_{from_i}:

$$CE_{usr} = \frac{1}{\sum_{t_i \in T} d_i z_{from_i} p_i} \tag{17}$$

As verified in Section 3.6 of my dissertation, the performance testing method presented above is a reasonable alternative of the ad-hoc performance testing method for two reasons. First, unlike the ad-hoc method, it is capable of calculating the worst-case number of messages the SUT is able to process within a second. Furthermore, when given the same amount of time for testing, the proposed method is capable of calculating the expected number of messages the SUT is able to process within a second with a higher precision (i.e. with a lower deviation).

4.3 Worst-Case Performance Correction of Communicating Systems

If the performance test finds that $CW_{usr} < CWR_{usr}$ that is, the worst-case number of messages that the SUT is able to serve within a second, is lower than the worst-case number of messages that the SUT should be able to serve within a second, then the performance of the SUT should be augmented to a level at which, it fulfills this requirement. Increasing the number of messages that the SUT is able to process within a second in worst case, is achieved by reducing its transition delays. Each transition delay is reducible by predefined amounts, which may vary from transition to transition. Furthermore, each transition delay reduction has a cost.

The methods presented in the following, aim at increasing the worst-case number of messages the system is able to serve within a second to the desired level, at minimal cost. From now on, this problem is referred to as the worst-case performance correction problem.

Thesis 3 *I have proven that the worst-case performance correction problem is NP-complete. I have formulated the problem as an integer linear program. Furthermore, I have proposed a heuristic algorithm for solving the worst-case performance correction problem with logarithmic cost functions.*

I have formulated the worst-case performance correction problem as follows:

Given are the set $T = \{t_i\}$ of transitions, and the set $C = \{C_i\}$ of cycles, each cycle $C_i = \{t_j\}$ being a set of transitions, and each transition t_j having a delay value d_j. Given are a positive number CWR_{usr}, a positive number K, assigned to each transition t_i a variable (a so-called correction factor) $0 < q_i \leq 1$, and a set $Q_i = \{q_{ij}\}$, where $q_{ik} < q_{i(k+1)}$ for each $k = 1, \ldots, |Q_i| - 1$, and $q_{i|Q_i|} = 1$. Furthermore for each transition, given is a monotonic decreasing function $Cost_i(x)$ for which $Cost_i : (0, 1] \rightarrow \mathbb{R}^+$, $Cost_i(1) = 0$. The question to be answered is as follows: Is it possible to choose the value of each q_i so that $\exists(q_{ij} \in Q_i) : q_i = q_{ij}$, and the following two inequalities are true?

$$\forall(c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j \leq \frac{|c_i|}{CWR_{usr}} \tag{18}$$

$$\sum_{i:t_i \in T} Cost_i(q_i) \leq K \tag{19}$$

In the above definition, q_i is a factor representing the reduction of d_i. The reduced delay of t_i is $d_i q_i$. $Cost_i(q_i)$ is the cost of the delay reduction of transition t_i. $\forall(i : t_i \in T) : Cost_i(1) = 0$, because if $q_i = 1$, the delay of t_i is not reduced, and its delay reduction does not cost anything. Finally, K is an upper bound for the cost of correcting the delays of all transitions. Formula 18 expresses that after the delay correction, the system has to meet Formula 13, while Formula 19 expresses that the total cost of delay correction must not exceed K.

Thesis 3.1 *[J3, J4] I have proven the NP-completeness of the worst-case performance correction problem.*

I proved the NP-completeness of the worst-case performance correction problem by reducing an arbitrary instance of the NP-complete knapsack problem in Section 4.2 of my dissertation [27]. Since the problem is NP-complete, an efficient way to solve it, is formulating it as an integer linear program and solving this integer linear program.

Thesis 3.2 *[J3, J4] For finding the optimal solution of the worst-case performance correction problem, I have formulated it as an integer linear program.*

The ILP formulation of the worst-case performance correction problem is the following binary linear peogram, where $cst_{ij} = Cost_i(q_{ij})$:

Minimize:

$$\sum_{i:t_i \in T} \sum_{j=1}^{|Q_i|} s_{ij} cst_{ij} \tag{20}$$

Subject to:

$$\forall(i : t_i \in T) : \sum_{j=1}^{|Q_i|} s_{ij} = 1 \tag{21}$$

$$\forall c_i \in C : \sum_{j:t_j \in c_i} d_j \sum_{k=1}^{|Q_i|} s_{jk} q_{jk} \leq \frac{|c_i|}{CWR_{usr}} \tag{22}$$

$$\begin{gathered} \forall(i : t_i \in T) : \forall(j = 1, 2, \ldots, |Q_i|) : \\ s_{ij} \in \{0, 1\} \end{gathered} \tag{23}$$

The unknown variables the values of which have to be found when solving the binary program are the s_{ij} variables. The value of each s_{ij} has to be set to 0 or 1 (Equation 23). Variables s_{ij}, where $j = 1, \ldots, |Q_i|$ are used for selecting the correction factor of transition t_i. As a solution of the BLP above, for each transition t_i, there is exactly one s_{ij} variable the value of which is 1. All the other s_{ij} variables belonging to t_i are set to 0 (consequence of Equations 21 and 23). If the value of s_{ij} is 1 then $q_i = q_{ij}$ and thus, correcting the delay of t_i costs $Cost_i(q_{ij})$. On the left side of Inequality 22, $\sum_{k=1}^{|Q_i|} s_{jk} q_{jk}$ equals correction factor q_i of transition t_i. Thus, Inequality 22 means that the corrected delay of each cycle c_i has to be lower than or equal to $\frac{|c_i|}{CWR_{usr}}$ (this corresponds to Inequality 18). Finally, in the objective function (Formula 20), $\sum_{j=1}^{|Q_i|} s_{ij} cst_{ij}$ equals $Cost_i(q_i)$ (the cost of correcting the delay of transition t_i). The value of $Cost_i(q_i)$ is chosen from set $\{cst_{ij} | j = 1, \ldots, |Q_i|\}$ by the appropriate s_{ij} variable set to 1. Thus, the objective function expresses that the total cost of correcting the transition delays should be minimal.

The time needed to solve the above formulated BLP can be huge thus, I have created a heuristic algorithm for solving the worst-case performance correction problem.

Thesis 3.3 *[J3, J4] I have proposed a heuristic algorithm for solving the worst-case performance correction problem in the case, where the cost functions of transition delay reduction are logarithmic.*

The algorithm is optimized for the case when $Cost_i(x) = -\gamma_i \log_a x$, where $\gamma_i > 0$ is a constant assigned to transition t_i. Algorithm 3 shows how my heuristic method works. In the algorithm, r is the so-called refreshing granularity.

The algorithm first sets the value of each correction factor q_i to its minimal value q_{i1} and then it runs iterations and increases each q_i to its next smallest legal value (i.e. from q_{ik} to $q_{i(k+1)}$) more or less frequently, until no further correction factor increasement is possible without violating any instances of Inequality 18. Before the first iteration and upon each increasement of q_i, the algorithm sets the value of α_i, which is the number of iterations that have to pass until the next increasement of q_i. The key step of the algorithm is this latter one that is, determining how many iterations have to pass until the next increasement of each q_i in order to keep the cost of delay reduction minimal.

Algorithm 3: Heuristics for solving the worst-case performance correction problem

input : T, C, CWR_{usr}, r, $\{Q_i | i : t_i \in T\}$, $\{\gamma_i | i : t_i \in T\}$

output: $\bigcup_{i:t_i \in T} \{q_i\}$

foreach $i : t_i \in T$ **do**
  $clist_i := \{j | t_i \in c_j\}$
foreach $i : t_i \in T$ **do**
  $q_i := q_{i1}$
if $\exists (c_i \in C) : \sum_{j:t_j \in c_i} d_j q_j > \frac{|c_i|}{CWR_{usr}}$ **then**
  return "unsolvable";
foreach $i : t_i \in T$ **do**
  $current_i := 1$
foreach $i : t_i \in T$ **do**
  if $|Q_i| > 1$ **then**
    $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
while $\exists i : (current_i < |Q_i|$
$\wedge (\forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}}))$ **do**
  foreach $i : t_i \in T$ **do**
    if $current_i < |Q_i|$ **then**
      $\alpha_i := \alpha_i - 1$
  foreach $i : t_i \in T$ **do**
    if $\alpha_i = 0 \wedge current_i < |Q_i|$
    $\wedge \forall j \in clist_i : (\sum_{k:t_k \in c_j \wedge k \neq i} q_k d_k) + q_{i(current_i+1)} d_i \leq \frac{|c_j|}{CWR_{usr}})$ **then**
      $q_i := q_{i(current_i+1)}$;
      $current_i := current_i + 1$
  if $current_i < |Q_i|$ **then**
    $\alpha_i := \left\lceil \frac{|clist_i| d_i r (q_{i(current_i+1)} - q_{i(current_i)})}{\gamma_i} \right\rceil$
return $\bigcup_{i:t_i \in T} q_i$;

In Section 4.5 of my dissertation, I have evaluated the time and cost efficiency of the proposed heuristic algorithm, a simple round-robin algorithm and the ILP, which could not be solved in many scenarios and thus, the optimum was substituted by a

rough lower estimate given by a linear program. According to the simulations, the proposed heuristic algorithm is more cost efficient than the round-robin algorithm, while unlike the ILP, its running time is reasonable.

5 Application of the Results

The results of the first thesis group can be applied for load distribution in a performance test environment, where the load generating software entities (VHs) that generate the load towards the SUT, have predefined starting times and execution times. Among many others, one concrete application area is when the emulated users communicating with the SUT are the VHs. According to simulations, the proposed ILP based method gets more efficient in means of TH utilization, as the number of testing hosts (THs) and the quotient of the average TH capacity and VH capacity decreases. The running time of the ILP based method can be however, unreasonable. The performance of the proposed heuristic algorithm gets better in means of TH utilization, as the quotient of the average TH capacity and the average VH capacity decreases and the deviation of VH capacities increases, without respect to the number of THs.

The automatic performance testing method presented in the second thesis group is applicable if the performance requirement of the SUT is the maximal number of request messages to be served within a second while serving a given number of users. Two examples for the application of this method are a web portal and a SIP proxy. [31] In the first case, the requests are the clicks of users, while in the second case, the requests are the request messages that have to be sent to the proxy, e.g. to initiate a call. The efficiency of the proposed performance testing method was compared to that of an ad-hoc performance testing method used in the industry. The experiments have shown that each worst-case performance measurement taken by the proposed method is lower than any of the ad-hoc measurements taken by the ad-hoc method. The experiments have also proven that the expected performance measurements taken by the proposed method are correct that is, their average equals the average of the measurements taken by the ad-hoc method. Furthermore, the experiments have shown that the deviation of values measured by the proposed method is significantly lower than the deviation of the ad-hoc measurements taken within the same amount of time. This means that the proposed method is more accurate than the ad-hoc method. According to the experiments, the above stands

for extreme cases as well, e.g. when each transition delay of the SUT has a normal or a uniform distribution.

The methods presented in the third thesis group can be used for improving the performance of the SUT in case it has failed the performance test, which was executed against it according to the method presented in the second thesis group. The proposed methods assume that a cost function is available for each state transition of the SUT, using which, the cost of a certain amount of delay reduction of the given transition can be calculated. The binary linear program formulated in this thesis group cannot be solved in many scenarios, due to the huge amount of time needed for solving it. However, unlike solving the BLP, the presented heuristic algorithm has a reasonable running time. According to simulations, the performance of the heuristic method gets better as the amount of the necessary correction ($\frac{CWR_{usr}}{CW_{usr}}$) increases.

Acknowledgments

First of all, I would like to thank my supervisors, Dr. Tibor Csöndes for all the time and energy he has put into consulting me and guiding my research work all through the past years, and Dr. Sarolta Dibuz for her useful advices and support.

I am also very grateful to Dr. Gyula Csopaki for inviting me to research the field of testing as an MSc student and for his continuous guidance and support.

Furthermore, I would like to thank my colleagues and roommates, Dr. Péter Babarczi, Gábor Árpád Németh, and Zoltán Novák for their unconditional help and the great work athmosphere they provided. I would also like to thank my colleague, József Ernő Marton, for his technical advices.

Also, I would like to thank the High Speed Networks Laboratory (HSNLab) for providing the technical and financial background for my research work, especially Dr. Róbert Szabó, Dr. Attila Vidács, and Dr. Sándor Molnár for their useful advices related to my publications, and Erzsébet Győri for all her help.

My acknowledgments would not be complete without also thanking my head of department Dr. Tamás Henk, Dr. Edit Halász, Prof. Gyula Sallai, Dr. Gusztáv Adamis, and Dr. Gábor Kovács for their valuable advices on finalizing my thesis.

At last, but not at least I would like to thank all my family members for supporting me all through my student years, for the exceptional studying oppurtunities they provided, and for helping me overcome all the obstacles arisen.

References

[1] **Utting, M and Legeard, B.** Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

[2] **Pretschner, A. and Philipps, J.** Methodological Issues in Model-Based Testing In *Model-Based Testing of Reactive Systems*, pp. 281–291, 2004.

[3] **Dorofeeva, R. and El-Fakih, K. and Maag, S. and Cavalli, A. R. and Yevtushenko, N.** FSM-based conformance testing methods: A survey annotated with experimental evaluation In *Information and Software Technology vol. 52 issue 12.*, pp. 1286–1297, 2010.

[4] **Lee, D. and Yannakakis, M.** Principles and Methods of Testing Finite State Machines – A Survey In *Proceedings of the IEEE vol. 84 issue 8*, pp 1090–1123, 1996.

[5] **Tretmans, J.** Specification Based Testing with Formal Methods: A Theory In *FORTE / PSTV 2000 Tutorial Notes*, 2000.

[6] **Chow, T.** Testing Software Design Modelled by Finite-State Machines In *IEEE Transactions on Software Engineering vol. 4 issue 3.*, pp. 178–187, 1978.

[7] **Fujiwara, S. and Khendek, F. and Amalou, M. and Ghedamsi, A.** Test Selection Based on Finite State Models In *IEEE Transactions on Software Engineering vol. 17 issue 6.*, pp. 591–603, 2002.

[8] **Sun, X. and Shen, Y. and Feng, C. and Lombardi, F.** Advanced Series in Electrical and Computer Engineering - Vol. 12, Protocol Conformance Testing Using Unique Input/Output Sequences, World Scientific Publishing, 1997.

[9] **Sabnani, K. and Dahbura, A.** A Protocol Test Generation Procedure In *Computer Networks and ISDN Systems vol. 15 issue 4.*, pp. 285–297, 1988.

[10] **Naito, S. and Tsunoyama, M.** Fault Detection for Sequential Machines by Transition-Tours In *Proc. 11th Ann. IEEE Internat. Symp. Fault-Tolerant Comput.*, pp. 238–243, 1981.

[11] **Kohavi, Z.** Switching and Finite Automata Theory, McGraw-Hill, 1978.

[12] **Lee, D. and Yannakakis, M.** Testing Finite-State Machines: State Identification and Verification In *IEEE Transactions on Computing vol. 43 issue 3.*, pp. 306–320, 1994.

[13] **Tretmans, G. J.** Test Generation with Inputs, Outputs, and Quiescence In *Proc. Tools and Algorithms for Construction and Analysis of Systems, Second International Workshop, TACAS*, pp. 127–146, 1996.

[14] **Kovacs, G. and Pap, Z. and Csopaki, G. and Tarnay, K.** Iterative Automatic Test Generation Method for Telecommunication Protocols In *Computer Standards & Interfaces vol. 28 issue 4.*, pp. 412–427, 2006.

[15] **Nemeth, G. A. and Pap, Z. and Kovacs, G. and Subramaniam, M.** A Bounded Incremental Test Generation Algorithm for Finite State Machines In *Proc. 19th IFIP Int. Conf. on Testing of Communicating Systems (TestCom/FATES)*, pp. 244–259, 2007.

[16] **Ghedamsi, A. and Bochmann, G.V.** Test Result Analysis and Diagnostics for Finite State Machines In *Proc. of the 12th International Conference on Distributed Computing Systems*, pp. 244–251, 1992.

[17] **Brinksma, E. and Tretmans, J. and Verhaard, L.** A Framework for Test Selection In *Proc. IFIP WG6.1 11th Int. Symp. on Protocol Specification, Testing, and Verification*, pp. 233–248, 1991.

[18] **El-Fakih, K., Yevtushenko, N., von Bochmann, G.** FSM-based Incremental Conformance Testing Methods In *IEEE Transactions on Software Engineering vol. 30 issue 7.*, pp. 425–436, 2004.

[19] **Ufuk Celikkan and Rance Cleaveland** Computing Diagnostic Tests for Incorrect Processes In *Proc. of the IFIP Symposium on Protocol Specification, Testing, and Verification*, pp. 263–278, 1992.

[20] **Kemper, P. and Kritzinger, P. and Bause, F. and Kabutz, H.** SDL and Petri Net Performance Analysis of Communicating Systems In *Proc. of the 15th International Symposium on Protocol Specification, Testing and Verification*, pp. 269–282, 1995.

[21] **Youness, O. S. and El-Kilani, W. S. and El-Wahed, W. F. A.** A Behavior and Delay Equivalent Petri Net Model for Performance Evaluation of Communication Protocols In *Computer Communications, vol. 31 issue 10.*, pp. 2210–2230, 2008.

[22] **El-Karaksy, M. R. and Nouh, A. S. and Al-Obaidan, A.** Performance Analysis of Timed Petri Net Models for Communication Protocols: A Methodology and Package In *Computer Communications vol. 13 issue 2.*, pp. 73–82, 1990.

[23] **Marsan, M. A. and Chiola, G. and Fumagalli, A.** Timed Petri Net Model for the Accurate Performance Analysis of CSMA/CD Bus LANs In *Computer Communications vol 10. issue 6.*, pp. 304–312, 1987.

[24] **Schieferdecker, I. and Stepien, B. and Rennoch, A.** PerfTTCN, a TTCN Language Extension for Performance Testing In *Proc. of the IFIP TC6 10th International Workshop on Testing of Communicating Systems*, pp. 21–36, 1997.

[25] **Dai, Z. R. and Grabowski, J. and Neukirchen, H.** TimedTTCN-3 - A Real-Time Extension for TTCN-3 In *Testing of Communicating Systems*, pp. 407–424, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[26] **Mingwei, X. and Jianping, W.** A formal approach to protocol performance testing In *Journal of Computer Science and Technology vol. 14 issue 1*, pp. 81–87, 1999.

[27] **Garey, M. R. and Johnson, D. S.** Computers and Intractability; A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., San Francisco, CA, 1990.

[28] **Coffman,Jr., E. G. and Garey, M. R. and Johnson, D. S.** Approximation algorithms for bin packing: a survey In *Approximation algorithms for NP-hard problems*, PWS Publishing Co., pp. 46–93, 1997.

[29] **Szabo, J. Z. and Csondes, T.** TITAN, TTCN-3 test execution environment In *Infocommunications Journal vol. 57. issue 1*, pp. 27–31, 2007.

[30] **Hoffman, K. and Kunze, R** Linear Algebra, Prentice Hall, 2nd Edition, pp. 161–162, 1971.

[31] **RFC 3261** SIP: Session Initiation Protocol, IETF, 2002.

Publications

Journal papers

[J1] **Erős, L. and Bozóki, F.** Test Component Assignment and Scheduling in a Load Testing Environment In *Periodica Polytechnica vol. 52 issue 3–4*, pp. 145–152, 2008.

[J2] **Erős, L. and Csöndes, T.** Model-Driven Black Box Performance Testing of Communicating Systems In *Journal of Universal Computer Science*, under review, 2011.

[J3] **Erős, L. and Csöndes, T.** Model-Driven Performance Correction of Communicating Systems In *Computing and Informatics*, under review, 2011.

[J4] **Erős, L. and Csöndes, T.** Model-Driven Diagnostics of Underperforming Communicating Systems In *Acta Cybernetica*, under review, 2011.

[J5] **Pernek, Á. and Erős, L. and Csöndes, T.** Kommunikáló rendszerek teljesítménytesztelése *Híradástechnika vol. 65 issue 7–8.*, pp. 28-32, 2010.

[J6] **Erős, L. and Bozóki, F.** Refactorisation Methods for TTCN-3 *Acta Polytechnica, Czech Technical University in Prague vol. 47 issue 4–5.*, pp. 33–37, 2007.

Conference papers

[C1] **Erős, L. and Csöndes, T.** Test Component Assignment in a Performance Testing Environment In *Proc. IEEE SoftCom*, pp. 399-403, Split-Dubrovnik, Croatia, 2008.

[C2] **Erős, L. and Csöndes, T.** An Automatic Performance Testing Method Based on a Formal Model for Communicating Systems In *Proc. 18th IEEE International Workshop on Quality of Service*, Paper No. 1569284789, Beijing, China, 2010.

[C3] **Bozóki, F. and Erős, L.** Refactoring Test Data Structures In *Proc. IEEE ConTel 2007: 9th International Conference on Telecommunications*, pp. 123-130, Zagreb, Croatia, 2007.

[C4] **Bozóki, F. and Erős, L.** Eliminating the Redundancy of TTCN-3 Sources In *Proc. TRANSCOM 2007: 7th European Conference of Young Research and Science Workers*, pp. 47-50, Zilina, Slovakia, 2007.

[C5] **Wu-Hen-Chang, A. and Adamis, G. and Erős, L. and Kovács, G. and Csöndes, T.** A New Approach in Model-Based Testing: Designing Test Models in TTCN-3 In *Proc. SDL 2011: 15th International Conference on System Design Languages*, pp. 90-105, Toulouse, France, 2011.